Bachelor's Thesis (Academic Year 2021)

Qubit Allocation For Distributed Quantum Computing

Keio University, Faculty of Environment and Information Studies
Makoto Nakai

卒業論文要旨 - 2021 年度 (令和 3 年度)

分散量子計算のための量子ビット割り当て

量子コンピュータは従来のコンピュータが現実的な時間内に解けないことが示されている問題の一部のの解より少ない時間計算量で得られることが理論的に示されている。大規模な量子計算を行うアプローチとして大規模かつ単一の量子プロセッサを構築するアプローチと複数の量子プロセッサ上で分散的に行うアプローチの 2 つが提案されているが、後者の方が量子プロセッサに要求される量子ビット数とエラー率が低いことから、より現実的であると考えられている。しかし、実際に分散量子計算を行う方法、特に量子回路のプログラム上の各量子ビットをどの量子プロセッサ上で実行するかを決定する方法について議論してされていない。本研究では量子回路全体の実行時間の短縮を目的とした、量子ビット割り当て問題に対する近似最適化アルゴリズムを提案し、数値シミュレーションで無作為の量子ビット割り当てに比べて、本研究が提案するアルゴリズムで最適化した量子ビット割り当てを採用した方が実行時間が短くなることをシミュレーションで実証した。

キーワード:
1. 量子計算, 2. 分散システム, 3. 仕事割り当て問題, 4. 擬似焼きなまし法

慶應義塾大学 環境情報学部 4 年
中井 慎

Abstract of Bachelor's Thesis - Academic Year 2021

Qubit Allocation For Distributed Quantum Computing

Quantum computers are theoretically capable of solving some intractable problems for classical computers including the factoring problem and the digital quantum simulations in a polynomial time, and large-scale quantum computing is the key to perform computation that even a supercomputer cannot handle. Two approaches for large-scale quantum computing have been proposed. One is to build a single large quantum processor, and the other is to perform quantum computing over more than one quantum processors that are connected via communication links. The later approach is considered as more practical because it requires less number of qubits and error rate, which are the two most challenging aspects of building a hardware for large-scale quantum processor.

However, only few works have investigated the procedure to perform distributed quantum computing in the real world, especially how to convert a user program which includes the quantum circuit to the executable form onto distributed quantum computers.

This work proposes the heuristic optimization algorithm for qubit allocation for distributed quantum computing which aim to shorten the total execution time of the given quantum circuit, and demonstrated that the qubit allocation by this algorithm achieve reduction of the total execution time compared to the random qubit allocation by numerical simulation.

Keywords :

1. Quantum computing, 2. Distributed system, 3. Task allocation, 4. Simulated Annealing

Keio University, Faculty of Environment and Information Studies

Makoto Nakai

Contents

List of Figures

List of Tables

List of Code Listings

Chapter 1

Introduction

1.1 Background

Quantum computers are theoretically capable of solving some intractable problems for classical computers including the factoring problem and the digital quantum simulations in a polynomial time, and large-scale quantum computing is the key to perform computation that even a supercomputer cannot handle. Two approaches for large-scale quantum computing have been proposed. One is to build a single large quantum processor, and the other is to perform quantum computing over more than one quantum processors that are connected via communication links. The later approach is considered as more practical because it requires less number of qubits and error rate, which are the two most challenging aspects of building a hardware for large-scale quantum processor.

However, unlike the previous works about the system for quantum computing on a single quantum processor, only few works have investigated the procedure to perform distributed quantum computing in the real world, especially how to convert an user program which includes the quantum circuit to the executable form onto distributed quantum computers.

1.2 Research Contribution

The main contribution of this project is the heuristic optimization algorithm for qubit allocation problem for distributed quantum computing, which determines which qubits and associated quantum gates will be executed on which quantum processor. This work adopts the total execution time as the optimization criteria, because the algorithm for qubit allocation with emphasis on total fidelity of the output quantum state can be solved by the existing quantum compilation techniques.

1.3 Thesis Structure

This thesis is constructed as follows.

Chapter 2 provides the fundamentals knowledge of quantum information, distributed system, and distributed quantum computing. In chapter 3 provides the previous researches in the fields of distributed quantum computing that are directly related to this work. Chapter 4 describes the details of the qubit allocation problem for distributed quantum computing and propose its heuristic solution. Chapter 5 explains the setting of the experiments and their results. Chapter 6 discusses the validity of the proposed approach. Chapter 7 provides a summary of this research and describes the future development of the research.

Chapter 2

Background

2.1 Quantum Computing

In this chapter, the author will provide fundamental knowledge about quantum computing, including the concepts of quantum bits, quantum gates, quantum circuit and quantum compilation.

2.1.1 Quantum Bit

A classical bit has two different states, which are 0 and 1. Instead, those of a quantum bit (or **qubit** in short) are $|0\rangle$ and $|1\rangle$, each of which can be described as a vector.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The state of a single qubit $|\psi\rangle$ can be described as follows.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \, (\alpha, \beta \in \mathbb{C}, |\alpha|^2 + |\beta|^2 = 1)$$

After the operation called measurement, the quantum state would be collapsed into either 0 or 1. The measurement probability of 0 is $|\alpha|^2$ and that of 1 is $|\beta|^2$. In other words, a single qubit can take both states probabilistically at the same time. For instance, a qubit can be

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

which can be 50% 0 and 50% 1.

2.1.2 Bloch sphere

Because $|\alpha|^2 + |\beta|^2 = 1$, the notation of a single qubit state can be represented like this.

$$|\psi\rangle = e^{i\gamma}(\cos\frac{\theta}{2} + e^{i\phi}\sin\frac{\theta}{2})(\gamma, \phi, \theta \in \mathbb{R})$$

Because $e^{i\gamma}$ is just a global state, it can be ignored and the same state can be rewritten like this.

$$|\psi\rangle = \cos\frac{\theta}{2} + e^{i\phi}\sin\frac{\theta}{2}(\phi, \theta \in \mathbb{C})$$

Because the equation above has two parameters, any pure single qubit state can be considered as a point on the surface and its geometric representation is called **Bloch sphere**.

Figure 2.1: Bloch sphere

2.1.3 Multi-Qubit State

The quantum state for multi-qubits is a **tensor product**of a state vector of each qubit. The general notation of two qubit state is

$$|\psi\rangle = (\alpha|0\rangle + \beta|1\rangle) \otimes (\gamma|0\rangle + \delta|1\rangle)$$

$$= \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle$$

$$(\alpha, \beta, \gamma, \delta \in \mathbb{C}, |\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1)$$

For example, the state $|00\rangle$ is equal to

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

However, some quantum states such as

$$|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

cannot be decomposed into quantum state of each qubit. These special quantum states are called **entangled** states.

2.1.4 Quantum Gates

In this section, the author will talk about ”logical gates” for quantum computers, which are called **quantum gates**.

I gate

I gate is equal to the 2x2 identity matrix, which is

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

For example,

$$I|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$$

$$X|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

X gate

X gate flips the logical value of a qubit.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

For example,

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

$$X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$$

Y gate

Y gate flips the logical value of a qubit and add an imaginary number.

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

For example,

$$Y|0\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ i \end{bmatrix} = i|1\rangle$$

$$Y|1\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -i \\ 0 \end{bmatrix} = -i|0\rangle$$

Z gate

Z gate flips the phase of $|1\rangle$

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

For example,

$$Z|0\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle$$

$$Z|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} = -|1\rangle$$

H gate

H gate creates superposition.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

For example,

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$H|1\rangle = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Rx gate

An Rx gate rotate the given quantum circuit on the x-axis of the Bloch sphere.

$$Rx(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$$

For example,

$$Rx(\theta)|0\rangle = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \cos\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} \end{bmatrix} = \cos\frac{\theta}{2}|0\rangle - i\sin\frac{\theta}{2}|1\rangle$$

$$Rx(\theta)|1\rangle = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -i\sin\frac{\theta}{2} \\ \cos\frac{\theta}{2} \end{bmatrix} = -i\sin\frac{\theta}{2}|0\rangle + \cos\frac{\theta}{2}|1\rangle$$

Ry gate

An Ry gate rotate the given quantum circuit on the y-axis of the Bloch sphere.

$$Rx(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$$

CNOT gate

A CNOT gate involves two qubits, one is called **controlled qubit** and the other is called **target qubit**. If the controlled qubit is 1, the bit value of the target qubit is flipped.

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$CNOT_{0,1}|10\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = |11\rangle$$

$$CNOT_{0,1}|11\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = |10\rangle$$

Quantum gate for multi-qubit system

Just like quantum state of a multi-qubit system, the composited quantum gates is the tensor product of quantum gates that are applied on each qubit.
For example,

$$X_0 \otimes X_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Therefore,

$$X_0X_1|00\rangle = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = |11\rangle$$

$$X_1|00\rangle = I_0X_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

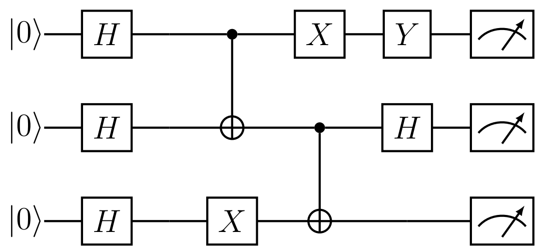
$$I_0X_1|00\rangle = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = |01\rangle$$

Measurement

If a person measure a single qubit, he would get either 0 or 1 and that operation completely destroys the quantum state. It will return 00, 01, 10, 11 in the case of two qubits.

2.1.5 Quantum Circuit

Here is the example of a quantum circuit.



Each horizontal line represents each qubit and the square boxes that contain alphabets mean single quantum gates. The sign which involves a vertical line means a CNOT gate, and the box on the most right side indicates measurement.

2.1.6 Bell state

In the chapter 1.3, the author mentions about a special type of quantum states called entangled states, and there are four specific quantum states called "Bell state", which are

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

$$|\Phi^-\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}}$$

$$|\Psi^+\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}}$$

$$|\Psi^-\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}}$$

2.2 Distributed Computing

Distributed computing is the study of distributed system, which is a collection of several processors that are connected via network in order to solve a problem whose scale is much larger than what an individual processor can handle. This type of computation involves message-passing between two physically separated processors to communicate and cooperate with each other by using pure HTTP, RPC, or message queues. [7]

2.2.1 Characteristics of Distributed System

Distributed system has the following characteristics [8], which are,

- No common physical clock

  This is the key feature of distributed system because the clock of each processor runs at a different rate, so no clock can keep synchronized even after a single physical clock cycle. Instead, distributed system depends on logical clock, which is common time platform for the whole system.

- No shared memory

  Each processor in a distributed system has its own memory space, rather than the common physical memory. This feature indicates that a distributed system does not share its global state.

- Geographical separation

  The processors in a distributed system is located in different places, but they do not have to communicate with wide area network (WAN). Actually, the network of workstations (NOW) and the cluster of workstations (COW) are becoming increasing popular because companies can easily purchase cost-efficient, high-speed, and ready-made processors.

- Autonomy and heterogeneity

  A distributed system can work together even if it contains various processors which have different size, speed and operating system as long as they cooperate with one another. This situation is regarded as ”loosely-coupled”.

2.2.2 Advantages Over Computation by a Single Processor

Performing computation in a distributed manner provides the following advantages. [8]

- Computation by more than one entity

  Applications such as money transfer (client-server) and reaching consensus among parties that are geographically separated (peer-to-peer) require information processing system that each processor can work together.

- Resource sharing

  Resource such as data in databases cannot be replicated because it is impossible or at least cost effective. Furthermore, allocating all the resource in just a single server is also not practical because the whole application would become unavailable if the server fails. In order to solve these potential problems, the whole dataset is usually partitioned into several servers so that it can achieve more rapid access and higher reliability.

- Access to geographically remote data and resources

  Copying the whole dataset to every site is not desirable due to not only its predicted high cost, as I mentioned in the previous section, but also too sensitive. Therefore, these large amount sensitive information, like user information collected by multinational cooperations are stored only in their central data centers and their oversea branches are only allowed to query them.

- Enhanced reliability

  Distributed system offers increased reliability due to its ability to replicate resource and achieve simultaneous execution of its given tasks. Also geographically distributed resources are highly unlikely crash or malfunction at the same time under normal circumstance. This eliability entails several aspects.

  – Availability

    Resource become accessible at all times

  – Integrity

    The value and state of the resources should be always correct, especially users get concurrent access to those resources.

  – Fault-tolerance

    Distributed system should be able to recover from its failure such as one of its server accidentally shutting down

- Increased performance / cost ratio

  By resource sharing and access to geographically distant data, the performance and cost ratio of distributed system will improve more than using special parallel machines, this is particularly true of the NOW (network of workstation) setting.

2.2.3 Task Allocation Problem

Here are some definitions of the task allocation problem in distributed system.[9] Given a distributed system $G =< V, E >$, where V is the set of nodes and E is the set of communication link between two different nodes, i.e. $\forall v_i.v_j \in V, \exists < v_i, v_j > \in E$. The set of the resources in a_i is R_{a_i} and that required by the task t is R_t.

1. $R_t \subseteq \bigcup_{\forall a_i \in A_t} R_{a_i}$
2. The objective should be either minimizing the execution time [10] or maximizing reliability [11]
3. The nodes in A_t can execute the allocated task under the constraint of the network structure $\forall a_i, a_j \in A_t \Rightarrow P_{ij} \subseteq E$ where P_{ij} denotes the path between a_i and a_j

Task allocation is known to be a NP-problem. [12]

2.2.4 Distributed Task Allocation Algorithms

Due to the fact that task allocation problem is classified as NP-hard, many works have proposed heuristic functions for distributed task allocations in various settings. Here, the author is going to introduce a simple objective function presented in the paper [13], which is this work is based on.

It is assumed that a multicomputer system consists of N heterogeneous processors with some amount of computational power and the size of memory. Also, the given program includes M communicating tasks, which is the nodes in the task interaction graph $G(V, E)$ (V is M communicating tasks and E is a set of communication relationship between two tasks).

The objective of this task allocation is to minimize the total execution time. In other words, the author has to come up with the optimal allocation A, which $A(i) = p$ indicates that the task i is allocated to the processor p and $TASKS_p$ is a group of tasks that are allocated to a processor p.

The total execution in the heterogeneous computing cluster is same as the execution time in the most heavily loaded processor. Two main types of costs should be considered.

One is the execution cost. The execution load in the processor p is the cost of processing all the tasks that are assigned to p for the allocation A. Suppose C_{ip} is the cost of processing the task i on the processor p, then the total execution cost on the processor p is

$$EXEC_p = \sum_{task \in TASKS_p} C_{task,p}$$

The other cost is the communication cost, which can be calculated by the following formula.

$$COMM_p = \sum_{task \in TASKS_p} \sum_{(i=j),(i,j) \in E, A(j) \neq p} d_{ij} * cc_{avg}$$

. dij is the data sent between two communicating tasks between i, j and cc_{avg} is the average amount of transferring a data unit through the network transmission media.

Therefore, the total cost for the processor p is

$$COST_p = EXEC_p + COMM_p$$

. Because the total execution time is equal to the execution of the most heavily loaded processor, the total execution time can be described as following.

$$COST = max\{COST_p | 1 \leq p \leq n\}$$

Therefore, the object function is

$$min\ COST$$

2.2.5 Models of Process Communication

There are two basic models of process communication.[13] One is *synchronous* communication, which the sender process blocks its execution until it receives "acknowledgement

" (or "ack" in short) from the receiver, which indicates that the receiver accepted the delivered message. In other words, the sender and receiver synchronize with each other, in order to adjust their timings to cooperate.

On the other hand, the other model, which is *asynchronous* execution does not require synchronization between the sender and the receiver. Therefore, the sender executes its following task right after it delivers its message to the receiver.

2.3 Distributed Quantum Computing

Performing quantum computation on a cluster of middle-sized quantum processors is an easier approach compared to build a large single processor due to its lower physical requirement to build each processor [13].
This chapter explains several components that consists of distributed quantum computing system.

2.3.1 Distributed Quantum Algorithms

Distributed version of Shor's algorithm[14] and Grover's algorithm[15], both of which takes remote CNOTs (which will be discussed in a later subsection) were presented in 2004 and 2012, respectively. However, the quantum circuits shown in these studies are physical ones, in other words, something that will be implemented directly on physical hardware, not something that users will implement in their programs.

2.3.2 Distributed Quantum Compiler

Quantum compiler is a software to optimize a quantum circuit defined in the given program to reduce the number of quantum gates and alleviate the effect of physical noise on that quantum state, and map that to the real hardware so that it will satisfy the connectivity constraint on that hardware [16]. Just like the case of a local quantum software, distributed quantum computer also needs its distributed version, which is called "distributed quantum compiler" in order to convert some CNOT gates into remote CNOTs and optimize both its execution time and the number of quantum gates in total, especially that of non-local operations. This work focuses on the methodology to optimize the total execution time over the distributed quantum computing setting.

2.3.3 Gate-Teleportation-Based Non-Local CNOT

Non-local operations are controlled operations between two qubits on two different processors there are three main approach to achieve them.

The first operation is the gate teleporation approach, and is also called non-local quantum gates [17] and telegate [18]. Here is the quantum circuit that achieve this operation.

Suppose that a person would like to apply a non-local CNOT between $|a_0\rangle$ on one processor and $|b_0\rangle$ on the other processor. In order to do this, he has to prepare a new bell pair between these two processors. ($|p_i\rangle$ in the following circuit diagram comes from p in $|\psi(psi)^+\rangle$)

Figure 2.2: Quantum circuit for a non-local CNOT gate

2.3.4 Quantum Teleportation

Unlike classical communication, quantum states cannot be just copied and transmit to other nodes due to the no-cloning theorem, which forbids duplication of any quantum state. However, a method called quantum teleportation[19] was proposed, which overcomes the restriction and allows sender to transmit single qubit state to a distant location.

This method requires both the single qubit state and a new Bell pair, and also the sender have to prepare two qubits and the receiver have to prepare one qubit. After applying a CNOT gate and an H gate in the figure above, the sender have to measure both qubits and send those measurement results over the classical network. After the receiver get those measurement results and apply some quantum gates if the measurement results of corresponding qubits on the sender's side are 1, in order to correct on the quantum state on the receiver's side.

Figure 2.3: Quantum circuit for quantum teleportation

2.3.5 Quantum-Teleportation-Based Non-Local CNOT

The second approach for performing a non-local CNOT gate is based on quantum teleportation mentioned in the previous section. This approach assumes that every quantum processor has something called "communication qubits", which is a qubit that serves for communication purpose, unlike those used for computation purpose, which are called "data qubits".

Figure 2.4: The full quantum circuit for a teledata non-local controlled-U gate

2.3.6 Data-Qubit-Swapping-Based Non-Local-CNOT

The last approach for the non-local CNOT gate aims to sort qubits in the quantum processors so that each CNOT gates would be executed on the neighboring processors. (the paper [20] assumes linear topology)

Figure 2.5: The quantum circuit before the data qubit swapping occurs

Figure 2.6: The quantum circuit after the data qubit swapping occurs

Algorithm 1 Algorithm for Data-Qubit Swapping

Input: n-qubit circuit layer L with mod(n, 4) = 0 and $\frac{n}{2}$ CNOTs
Output: layer L with each CNOT operating on neighbor qubits

function Sort(L)
    if $\exists CNOT(q_i, q_j)$ with $i, j \leq \frac{n}{2}$ **then**
        // $\exists CNOT(q_k, q_l)$ with $k, l > \frac{n}{2}$
        SWAP (q_{i+1}, q_j)
        SWAP(q_{k+1}, q_l)
        L = L $\setminus\{q_i,\ q_{i+1},\ q_k,\ q_{k+1}\}$
    else
        // $\exists CNOT(q_{\frac{n}{2}}, q_l)$ with $l > \frac{n}{2}$
        // and $\exists CNOT(q_i, q_{l-1})$ with $i < \frac{n}{2}$
        SWAP ($q_{\frac{n}{2}}$, q_{l-1})
        SWAP (q_i, $q_{\frac{n}{2}-1}$)
        L = L $\setminus\{q_{\frac{n}{2}-1},\ q_{\frac{n}{2}},\ q_{l-1},\ q_l\}$
    end if
    if $L \neq \emptyset$ **then**
        Sort(L)
    end if
end function

2.3.7 Quantum Processor

A quantum processor, which corresponds to individual processor in a classical distributed system, has several qubits and links between two qubits in a limited topology.

Figure 2.7: An example of the layout of a quantum processor

Usually, qubits in a current quantum processor are connected with a few neighboring qubits due to its physical restriction on a hardware. Also, the error rate of a CNOT gate is much higher than that of a single quantum gate.

2.3.8 Communication Link for Distributed Quantum Computing

Networking between two quantum processors need two types of communication links, which are classical links that transmit measurement results in the process of either telegate or teledata non-local CNOT gate, and quantum links that prepare create entanglement between the two qubits between the two neighboring quantum processors, or communication qubits of both processors.

Chapter 3

Related Works

3.0.1 Performance of An Interprocessor CNOT Gate

Both the execution time and required amount of resource for inter-node communication are important because it significantly affects the total execution time and the whole architecture of a distributed quantum computing system.

The work [19] proved that one bit of classical communication in each direction and one bell pair are sufficient for the non-local CNOT gate. It also proposes the optimal implementation of a non-local CNOT in terms of communication overhead and the number of required quantum gates.

The work [20] compared the performance between ”telegate” and ”teledata” approach in terms of various number of data qubits, communication qubits, and its network topology. It presents the fact that the teledata approach is faster than the telegate approach, and that decomposition of a quantum gate will improve the performance. It also shows that each node should have a few logical qubits and two communication qubits.

3.0.2 Minimization of the Number of Interprocessor Communication

Dividing the given quantum circuit into several fragments one of the main jobs for a distributed quantum compiler. In the process of partitioning the given quantum circuit, the distributed quantum compiler has to minimize the number of inter-processor communication in order to reduce the delay in the entire circuit execution.

The work [21] proposed an exhaustive-search-based algorithm to find a partition of the given quantum circuit with the minimum number of quantum teleportation between two quantum processors.

The work [22] applied the heuristic algorithm for a hypergraph partitioning problem in order to minimize the number of inter-processor communication, which can be applied to more than two quantum processors.

The work [23] tackled the minimization of interprocessor communication by using the genetic algorithm and demonstrated its advantage over random search over the search space.

The work [24] converted the given quantum circuit into a bi-partite graph and the gates and qubits are assigned to each part of the graph. Then, the graph was partitioned into K parts which minimizes the number of non-local CNOTs.

The work [25] used the Kernigrahn-Lin algorithm, which is one of the heuristic algorithms for the graph partitioning problem, to find arbitrary number of partition with the smallest number of interprocessor communication.

The work [26] proposed a new scheme for reducing interprocessor communication called window-based quantum circuit partitoning, or WQCP in short. This approach combined reduction of both telegate and teledata opportunities.

The work [26] suggested another algorithm for minimizing the interprocessor communication which consists of two phases by using the connectivity-matrix-based representation of a quantum circuit. In the first phase, it proposed two objective functions to minimize the number of non-local CNOTs and difference between the number of qubits in two partitioning. In the second phase, two heuristics were also proposed two other heuristics to minimize the number of quantum teleportation required.

3.0.3 Distributed Quantum Compiler

The work [27] discussed the design of a general-purpose, efficient, and effective distributed quantum computer. General purpose means no assumption about the given quantum circuit. Efficient means polynomial time complexity that grows polynomially with the number of qubits and linearly with the circuit depth. Effective assures a polynomial worst-case overhead in terms both depth of the compiled circuit, the number of entanglement generation.

This study also derived the analytical upper bound of the circuit depth both for the entanglement-based non-local operation and the data-qubit-swapping-based non-local operation.

The depth overhead of the entanglement-swapping-based strategy would be at most

$$\frac{n}{2}d_{es}$$

$$d_{es} = c_{le} + c_{bsm} + c_{cx}$$

.

On the other hand, the depth overhead of the data-qubit-swapping strategy is

$$\frac{n}{4}d_{qs} + d'_{qs}$$

$$d_{qs} = 3(c_{le} + c_{bsm} + c_{cx})$$

$$d'_{qs} = c_{le} + c_{cx}$$

n is the number of qubits, c_{le} is the number of layers required to perform the link entanglement, c_{bms} is that to perform entanglement swapping, c_{cx} is that to perform remote CNOTs. This study mentions that parameters c_{le}, c_{bms}, c_{cx} heavily depends on the underlying hardware architecture.

It also compare the performance of both strategies with the previous work [22]. It experimentally demonstrated that the entanglement-swapping-based strategy requires less number of layers for link generation, and the data-qubit-swapping-based strategy requires less circuit depth, on the worst network topology (the linear topology with one qubit on an each processor).

Chapter 4

Problem Definition and Proposal

4.1 Problem Definition and Proposal

This chapter explains a new problem about the qubit allocation in the distributed quantum computing system and propose the solution for that problem.

4.1.1 Problem Definition

Algorithms regarding distributed computing aim to achieve efficiency in terms of reduction in the total execution time and reliability compared to the case of execution of the same tasks on the single processor. In order to execute distributed computing, a collection of tasks should be allocated onto multiple processors limited by a certain network topology. These processors might have different properties such as their execution time and their memory size.

Previous works about qubit allocation on a single quantum processor tries to improve the fidelity of the output quantum state even in the presence of hardware noise, which is "reliability" for quantum computing on a current quantum processor. Those about quantum circuit distribution (or qubit allocation) for distributed quantum computing tries to minimize the number of interprocessor communication, in order to reduce the communication error, which is predicted to be worse than that in a local processor.

However, qubit allocation for distributed quantum computing to minimize the total execution time has not been investigated, even though this is the other optimization criteria for task allocation problem in the classical setting.

4.1.2 Formulation as An Optimization Problem

Suppose a distributed quantum computing system consists of N quantum processors connected via communication links. Each quantum processor has limited number of qubits and execution time.

A quantum circuit in the program consists of several qubits and M gates, including CNOTs which corresponds to an interaction graph $G(V, E)$. V represents a set of qubits and E represents set of two qubits involved in each CNOT gate. $q_i \in V$ is labeled by the qubit index, and $(control, target) \in E$ is labeled by control-target relationship of all the CNOT gates.

The problem is how to allocate each qubits in the given quantum circuit to which processor with varying execution time in order to minimize the total execution time. This problem can be formulated as an optimization problem, which requires a cost function, which is the value to either maximize or minimize to acquire the optimal solution.

4.1.3 Objective Function

Suppose A be the optimal assignment such that $A(q_i) = p_j$ if a qubit q_i in the given quantum circuit to a quantum processor p_j. Qubits allocated to a quantum processor p_j is denoted as $qubits_j$, single qubit gates allocated to a qubit q_i is $gates_i$ and the execution time on a quantum processor q_j is $time_j$.

The cost of executing all single-qubit gates on a qubit q_i on a quantum processor p_j is

$$\sum_{gate \in gates_i} time_j$$

Therefore, the cost of executing all the single-qubit gates on all the qubits allocated on quantum processor p_j is

$$GATECOST_j = \sum_{qubit \in qubits_j} \sum_{gate \in gates_{qubit}} time_j$$

Suppose a CNOT gate CNOT(control, target) involves two qubits, which are control $\in qubits_s$ and target $\in qubits_t$ and all the CNOT gates in a quantum processor p_j are denoted as $CNOTs_j$.

The communication cost in a quantum processor p_j is

$$COMMCOST_j = \sum_{\substack{CNOT(control,target) \in CNOTs_j \\ control \in q_s \\ target \in q_t}} pathlength(s, k)$$

$pathlength$ is the length of the path between the processor s and the processor t on the given network topology, and the processor j is same as at least either the processor s or the processor t.

Thus, the total cost on a quantum processor p_j is

$$COST_j = GATECOST_j + COMMCOST_j$$

Both execution of single qubit gates and communication with other processors affect the total execution time on each processor, and because the processor with the greatest cost will decide the total execution time on the whole distributed quantum system, the following value should be calculated.

$$MAXCOST = \max\{COST_j | 1 \leq j \leq N\}$$

Also, minimizing this value will reduce both the execution time for quantum gates execution and interprocessor communication, and the objective function of this problem is

$$min\, MAXCOST$$

4.1.4 Simulated Annealing

Simulated annealing is a heuristic algorithm which reaches to the global optimal solution in some cases [15][16]. Its idea comes from cooling the molten metal until it gains crystal structure, so it calculates the values of "temperature", which how long the optimization has been executed and "energy" which evaluates how close the current answer is to the optimal solution. The algorithm starts with high temperature and energy, and as the temperature becomes lower, the solution changes randomly and the combination and the answer after randomization process is accepted even its energy becomes higher than its previous answer in order to avoid being stuck in the local minimum.

Here is the pseudocode for the simulated annealing algorithm.

Algorithm 2 Simulated Annealing

Input: A random allocation A, temperature T, iteration number IterNum
Output: The optimal allocation A'

function SimulatedAnnealing(A, T, IterNum)
    A' = A
    for iter := 1 to IterNum **do**
        temp := T * (1− iter/IterNum)
        copyA ← copy(A')
        newA ← move(copyA)
        eng ← calc_energy(copyA)
        neweng ← calc_energy(newA)
        if accept_prob(eng, neweng, temp) > randomvalue(0, 1) **then**
            A' = NewA
        end if
    end for
    ↵ A'
end function

Algorithm 3 Finding a neighbor state

Input: Processor list P $\{P_0, P_1, \dots P_N\}$, initial allocation A $\{P_0 : qubits_0, P_1 : qubits_1, \dots P_n : qubits_n\}$
Output: New allocation A

function move
    $P_i \leftarrow$ a randomly selected processor
    $P_j \leftarrow$ another randomly selected processor
    $qindex_i \leftarrow$ a randomly selected qubit index from 0 to $len(qubits_i)$
    $qindex_j \leftarrow$ a randomly selected qubit index from 0 to $len(qubits_j)$
    $A[P_i][qindex_i], A[P_j][qindex_j] = A[P_j][qindex_j], A[P_i][qindex_i]$
    ↵ A
end function

Algorithm 4 Calculating the energy value

Input: initial allocation A $\{P_0 : qubits_0, P_1 : qubits_1, \ldots P_n : qubits_n\}$, a quantum gate list *gate_list* $\{gate_0, \ldots, gate_N\}$, an execution time list *time_list* $[time\_0, \ldots time_N]$, network topology N

Output: An energy value E

```
function CALC_ENERGY(A, gate_list)
    processor_list ← list(A.keys())
    gate_cost_list ← [0forprocessorinprocessor_list]
    comm_cost_list ← [0forprocessorinprocessor_list]
    for processor_id := 0 to len(processor_list) − 1 do
        for gate := gate_0 to gate_N do
            if gate.name ≠ CNOT and gate.index ∈ A[processor_id] then
                gate_cost_list[processor_id] += time_list[processor_id]
            end if
        end for
    end for
    distance_matrix ← N.distance_matrix
    for processor_id := 0 to len(processor_list) − 1 do
        for gate := gate_0 to gate_N do
            if gate.name = CNOT then
                if gate.index, gate.target_index ∈ A[processor_id] then
                    comm_cost_list[processor_id] += 0
                else if gate.index ∈ A[processor_id] then
                    for processor′_id := 0 to len(processor_list) − 1 do
                        if gate.target_index ∈ A[processor′_id] then
                            distance ← distance_matrix[processor_id][processor′_id]
                            comm_cost_list[processor_id] += distance
                        end if
                    end for
                else if gate.gate_index ∈ A[processor_id] then
                    for processor′_id := 0 to len(processor_list) − 1 do
                        if gate.index ∈ A[processor′_id] then
                            distance ← distance_matrix[processor′_id][processor_id]
                            comm_cost_list[processor_id] += distance
                        end if
                    end for
                end if
            end if
        end for
    end for
    for processor_id := 0 to len(processor_list) − 1 do
        gate_cost ← gate_cost_list[processor_id]
        comm_cost ← comm_cost_list[processor_id]
        cost_list[processor_id] ← gate_cost + comm_cost
    end for
    ↵ max cost_list
end function
```

Algorithm 5 Calculating the acceptance probability

Input: current energy value *cur_eng*, new energy value *new_eng*, current temperature *temp*

Output: an acceptance probability *prob*

function ACCEPT_PROB(cur_eng, new_eng, temp)
    if cur_eng $<$ new_eng **then**
        ↵ 1
    else
        ↵ $\exp\left(-(new\_eng - cur\_eng)/temp\right)$
    end if
end function

Chapter 5

Evaluation

In this chapter, the author investigates the efficiency of the allocation method proposed in the previous chapter, under the distributed quantum computing system with several processors with different number of qubits, different execution time, and the limited network topology. The following experiments compare the total execution time in each setting among the case of random allocation, the case when only the gate cost is optimized, the case when only the communication cost is optimized, and the case when both costs are optimized.

These experiments were performed by a distributed quantum computing simulator called HeqSim(Heterogeneous Quantum Computing Simulator) [28] that I have developed.

5.0.1 Experiment Settings

Here are the details of the setting of each experiment.

Table 5.1: The details of all experiment settings

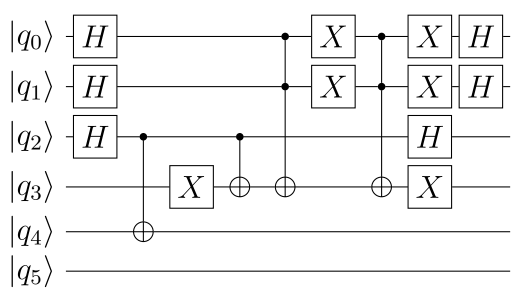
# of Experiment	# of Processors	# of total qubits	Network topology
1	2	6	linear
2	4	8	linear
3	4	8	ring
4	4	8	star
5	4	8	random

5.0.2 Experiment 1: Qubit Allocation to Two Quantum Processors

The first experiment is the simplest one, which is allocation of qubits onto two quantum processors that are connected with each other, and each processor has different number of qubits and gate execution time.

Figure 5.1: The details of each quantum processor

The author executed the following circuit on these two quantum processors. This circuit is a quantum circuit for solving 6-qubit Simon's problem and also a part of the set of quantum circuits that can be used for benchmarking purpose[20].



The author compared the total execution time of three different allocation cases, which are when only the gate execution cost is optimized, when only the communication cost is optimized, and when both costs are optimized. Here is the outcome of this experiment. (The value in each case is the average value of 10 different execution)

Figure 5.2: Result of The Experiment1

As you can see in the figure above, the case of random allocation was outperformed by the three other cases, and actually the case when gate execution cost was optimized yielded the best result.

5.0.3 Experiment 2: Qubit Allocation to Four Quantum Processors (linear topology)

5.0.4 Experiment 3: Qubit Allocation to Four Quantum Processors (ring topology)

5.0.5 Experiment 4: Qubit Allocation to Four Quantum Processors (star topology)

5.0.6 Experiment 5: Qubit Allocation to Four Quantum Processors (random topology)

Chapter 6

Discussion

6.1 Discussion

Although, as shown in the last chapter, the case when only the gate execution time is optimized yielded the best result, both the case when gate execution cost is optimized and both costs are optimized reached to the most optimal qubit allocation, which are (1,3) and (0, 2, 4, 5). The difference between the total execution times between two cases happened due to the nature of parallel execution in my simulator, which is that each gate is executed in not rigorously sequential order and the delay of communication between two threads, which emulate quantum processors.

Chapter 7

Conclusion

7.1 Conclusion

In this research, I proposed a method for generating traffic matrices based on current Internet methods, and implemented it in a Quantum Internet simulator.

Bibliography

[1] Yin Zhang, Matthew Roughan, Nick Duffield, and Albert Greenberg. Fast accurate computation of large-scale ip traffic matrices from link loads. *ACM SIGMETRICS Performance Evaluation Review*, 31(1):206–217, 2003.

[2] 辻 舛二. **電話トラフィック理論とその応用**. 電子通信学会, 1954.

[3] Walter Willinger and Vern Paxson. Where mathematics meets the internet. *Notices of the AMS*, 45(8):961–970, 1998.

[4] Will Koehrsen. The poisson distribution and poisson process explained. `http://www.soumu.go.jp/menu_news/s-news/01tsushin02_02000072.html`, 2019. Accessed: 2021-12-18.

[5] 福田健介 et al. ネットワークトラフィックの自己相似性とその生成モデル. **情報処理**, 45(6):603–609, 2004.

[6] 上田浩, 奈須野裕, 岩谷幸雄, 五十嵐隆治, 木下哲男, et al. 確率過程による lan トラヒックのモデル化における一考察. **情報処理学会論文誌数理モデル化と応用** *(TOM)*, 48(SIG2 (TOM16)):167–174, 2007.

[7] Matthew Roughan. Internet traffic matrices. `https://roughan.info/project/traffic_matrix/`, 2017. Accessed: 2021-12-18.

[8] University of TARTU Institute of Computer Science. Gravity models. `https://courses.cs.ut.ee/2011/graphmining/Main/GravityModels`, 2011. Accessed: 2021-12-22.

[9] Nabili Benameur and JW Roberts. Traffic matrix inference in ip networks. *Networks and Spatial Economics*, 4(1):103–114, 2004.

[10] Matthew Roughan. Simplifying the synthesis of internet traffic matrices. *ACM SIGCOMM Computer Communication Review*, 35(5):93–96, 2005.

[11] Bernard Fortz, Jennifer Rexford, and Mikkel Thorup. Traffic engineering with traditional ip routing protocols. *IEEE communications Magazine*, 40(10):118–124, 2002.

[12] Vern Paxson and Sally Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on networking*, 3(3):226–244, 1995.

[13] Matthew Roughan, Mikkel Thorup, and Yin Zhang. Traffic engineering with estimated traffic matrices. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, pages 248–258, 2003.

[14] Matthew Roughan, Albert Greenberg, Charles Kalmanek, Michael Rumsewicz, Jennifer Yates, and Yin Zhang. Experience in measuring internet backbone traffic variability: Models metrics, measurements and meaning. In *Teletraffic Science and Engineering*, volume 5, pages 379–388. Elsevier, 2003.

[15] Ryosuke Satoh, Michal Hajdušek, Naphan Benchasattabuse, Shota Nagayama, Kentaro Teramoto, Takaaki Matsuo, Sara Ayman Metwalli, Takahiko Satoh, Shigeya Suzuki, and Rodney Van Meter. Quisp: a quantum internet simulation package. *arXiv preprint arXiv:2112.07093*, 2021.

[16] Takaaki Matsuo, Clément Durand, and Rodney Van Meter. Quantum link bootstrapping using a ruleset-based communication protocol. *Physical Review A*, 100(5):052320, 2019.

[17] Takaaki Matsuo, Takahiko Satoh, Shota Nagayama, and Rodney Van Meter. Analysis of measurement-based quantum network coding over repeater networks under noisy conditions. *Physical Review A*, 97(6):062328, 2018.

[18] Takaaki Matsuo. Simulation of a dynamic, ruleset-based quantum network. Master's thesis, Keio University, Graduate School of Media and Governance, 2019.

[19] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[20] Matthew Roughan. How i learned to stop worrying and love traffic matrices. `https://roughan.info/talks/tma_summer_school.pdf`, 2016. Accessed: 2021-12-22.

[21] Paul Tune, Matthew Roughan, H Haddadi, and O Bonaventure. Internet traffic matrices: A primer. *Recent Advances in Networking*, 1:1–56, 2013.

[22] Paul Tune and Matthew Roughan. Spatiotemporal traffic matrix synthesis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 579–592, 2015.

[23] Grigorios Kakkavas, Michail Kalntis, Vasileios Karyotis, and Symeon Papavassiliou. Future network traffic matrix synthesis and estimation based on deep generative models $\gg$. In *30th International Conference on Computer Communication and Networks (ICCCN)*, 2021.

[24] Paul Tune and Matthew Roughan. Maximum entropy traffic matrix synthesis. *ACM SIGMETRICS Performance Evaluation Review*, 42(2):43–45, 2014.

[25] Yin Zhang, Matthew Roughan, Walter Willinger, and Lili Qiu. Spatio-temporal compressive sensing and internet traffic matrices. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 267–278, 2009.

[26] Will E Leland, Walter Willinger, Murad S Taqqu, and Daniel V Wilson. On the self-similar nature of ethernet traffic. *ACM SIGCOMM Computer Communication Review*, 25(1):202–213, 1995.